Please write your Entry Numer and Name in each page.

Entry No.: **Name:**

COL 733 MAJOR EXAM

Semester II, 2021-2022

- *Please do not carry bag, phone, calculator, or any electronic device with you in the desk. Put your phone to silent mode or switch it off to avoid distraction.*
- *Keep your ID card at the desk.*
- *Max marks for each question is marked in [ ].*
- *Write only in the provided space, no rough sheets will be provided. Justify your answers.*

Duration: 2 hours **Total marks: 75**

For answering Q1 and Q2, refer to the handout material.

Q1 *[18 marks]* (CPU Virtualization): Let's say we wish to build a "trap-and-emulate" based virtual machine monitor for 8086. For each of the following instructions, describe whether the instruction needs to be privileged. Justify your answer.

a. *NOP (no operands) [2 marks]:* NOP (No Operands) causes the CPU to do nothing. NOP does not affect any flags.

b. *SAHF (no operands) [2 marks]*: SAHF (store register AH into flags) transfers bits 7,6, 4, 2 and 0 from register AH (see Figure 2-32) into SF, ZF, AF, PF and CF, respectively, replacing whatever values these flags previously had. OF, DF, IF and TF are not affected.

c. *POPF (no operands) [2 marks]:* POPF transfers specific bits from the word at the current top of stack (pointed to by register SP) into the 8086/8088 flags, replacing whatever values the flags previously contained (see figure 2-32). SP is then incremented by two to point to the new top of stack. PUSHF and POPF allow a procedure to save and restore a calling program's flags. They also allow a program to change the setting of TF (there is no instruction for updating this flag directly). The change is accomplished by pushing the flags, altering bit 8 of the memory-image and then popping the flags.

d. *CLI (no operands) [2 marks]:* CLI (Clear Interrupt-enablel flag) zeroes IF. When the interrupt-enable flag is cleared, the 8086 and 8088 do not recognize an external interrupt request that appears on the INTR line; in other words maskable interrupts are disabled. CLI does not affect any other flags.

e. *ADD destination, source [2 marks]:* The sum of the two operands, which may be bytes or words, replaces the destination operand. Both operands may be signed or unsigned binary numbers. ADD updates AF, CF, OF, PF, SF, and ZF. *Note: Destination and source can be a general register or a logical memory location; the source can also be a constant. Example usages (from Table2-21 of the manual):*

ADD destination,source

register, register	ADD CX, DX
register, memory	ADD DI, ALPHA
memory, register	ADD ALPHA, CL
register, immediate	ADD CL, 2
memory, immediate	ADD ALPHA, 2

accumulator, immediate	ADD AX, 200

f. *JMP target [2 marks]:* JMP unconditionally transfers control to the target location. Unlike a CALL instruction, JMP does not save any information on the stack, and no return to the instruction following the JMP is expected. Like CALL, the address of the target operand may be obtained from the instruction itself (direct JMP) or from memory or a register referenced by the instruction (indirect JMP).

An intrasegment direct JMP changes the instruction pointer by adding the relative displacement of the target from the JMP instruction. If the assembler can determine that the target is within 127 bytes of the JMP, it automatically generates a two-byte form of this instruction called a SHORT JMP; otherwise, it generates a NEAR JMP that can address a target within +32k. Intrasegment direct JMPS are self-relative and are appropriate in position-independent (dynamically relocatable) routines in which the JMP and its target are in the same segment and are moved together.

Note: 8086 also allows "intersegment JMP" which also changes CS (Code Segment) register. For this problem, we will just assume that only intrasegment JMPs as described above are allowed.

g. *NOT destination [2 marks]:* NOT inverts the bits (forms the one's complement) of the byte or word operand. *Note: Destination can be a general register or a logical memory location (from Table2-21 of the manual):*

NOT destination

register	NOT AX
memory	NOT ALPHA

h. *PUSH source [2 marks]:* PUSH decrements SP (the stack pointer) by two and then transfers a word from the source operand to the top of stack now pointed to by SP. PUSH often is used to place parameters on the stack before calling a procedure; more generally, it is the basic means of storing temporary data on the stack. *Note: Source can be either a general register or a logical memory location.*

i. *MOV destination, source [2 marks]:* MOV transfers a byte or a word from the source operand to the destination operand. *Note: Source and destination can be general registers, segment registers, or a logical memory location. Source can also be constants. Following shows some example from Table2-21 of the manual.*

memory, register	MOV ALPHA, AL
register, memory	MOV, AX, ALPHA
register, register	MOV AX, CX
register, constant	MOV CL, 2

Entry No.: **Name:**

seg-reg, register	MOV ES, CX
seg-reg, mem	MOV DS, ALPHA
reg, seg-reg	MOV BP, SS

Q2 [16 marks] (Memory virtualization): Describe some schemes for doing memory virtualization.
Hints: You may assume that the hardware has more memory available that 1MB.

a. [4 marks] Paravirtualization: Guest OS is aware that it is being virtualized.

b. [4 marks] Hardware-assisted full virtualization: Guest OS is unaware of being virtualized, hardware assists the hypervisor in virtualization;

c. [4 marks] Software-based full virtualization: Guest OS is unaware of being virtualized, hypervisor is isolating guests from each other completely in software.

d. [4 marks] Compare their pros and cons.

Q3 [5 marks] (Raft heterogenous leader election): Let's say that the Raft replicas are heterogenous i.e, with differing hardware. Since Raft uses a "strong leader", how can we tweak the algorithm such that a more powerful machine (with a faster CPU/network card/disk) is more likely to become the leader. Also, justify why your new leader-election algorithm is safe+ live, given that any machine can crash arbitrarily.

Q4 [4 marks] (Raft state persistence): Mark persistent and non-persistent state in Raft. Give reasons why something should be persistent or why it is ok to be non-persistent? Assume that we are not storing the state machine state in persistent storage i.e, we are not creating any checkpoints.

a. [1 marks] currentTerm

b. [1 marks] votedFor

c. [1 marks] log[]

d. [1 marks] lastApplied

Entry No.: **Name:**

Q5 [6 marks] (Zookeeper): Describe how to implement ZooKeeper on top of Raft. Use Raft variable names (votedFor, lastApplied, commitIndex, etc.) to be precise about your implementation. Remember that ZooKeeper provides linearizable writes + client FIFO and allows reading directly from replicas. Prove why your implementation provides linearizable writes + client FIFO.

Q6 [6 marks] (Sharding): Let's say we wish to manage N (>>3) shards over 3 storage nodes. These shards can be anything: ranges of keys in a key-value store, different tables in a database, or some other concurrent data structures. We would like reads and writes within each of these shards to be linearizable, but, we need not have linearizability across shards.

Devise a scheme to do so (Justification of your scheme contains points, the scheme itself will not carry any point).

a. [1 mark] Justify how your scheme provides linearizability guarantees within a shard.

b. [1 mark] Justify how your scheme is fault-tolerant.

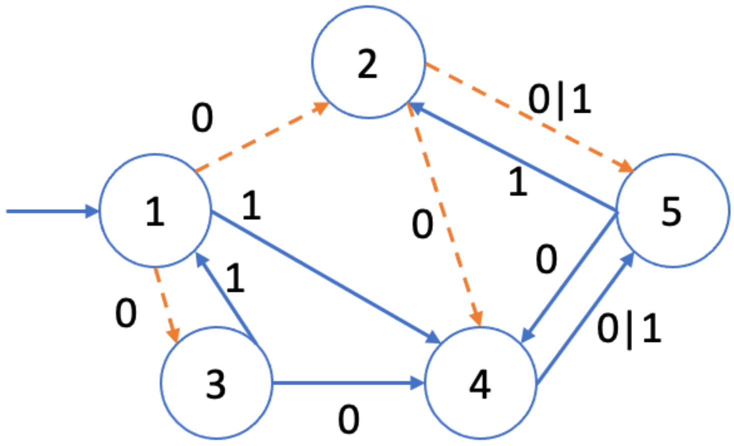
c. [1 mark] Discuss the availability of your scheme: in what failure conditions does your scheme continue to be available (continue to server read/write requests) and when does your system become unavailable. Justify your answer.

d. [1 mark] Assuming read-write traffic to each shard is equal, how does your scheme provide load balancing? Each storage node is doing equal work.

e. [2 marks] Let’s say that some of the shards can become hotter than others. Devise a decentralized scheme to dynamically rebalance the load among the servers, i.e, there is no separate master server overlooking the load and redistributing the load.

Q7 [10 marks] (Non-deterministic FSM): Raft is a mechanism for building replicated state machines. However, Raft's construction only allows deterministic state machines. How could you modify Raft to allow some non-determinism?

In particular, let's say Raft is managing a non-deterministic state machine as shown in the Figure below. The state machine is *mostly deterministic* with *a few* non-deterministic transitions as shown in the dashed orange color in the figure. There are only two types of requests clients can send concurrently: writing 0/1 or reading the current state (the current state is also just a small integer).



Justify that your modifications are able to preserve safety (linearizability) and liveness (makes progress as long as the majority nodes can talk to one another). For instance, the following history must not happen after arbitrary failures:

Client: read = state 1, write 0, read = state 2, read = state 3.

Entry No.: **Name:**

Q8: [10 marks] (IO virtualization): Following is a code-snippet from QEMU. qemu/hw/input/stellaris_input.c. Comments starting from "COL733" represents instructor comments.

```
/*
 * Gamepad style buttons connected to IRQ/GPIO lines
 *
 * Copyright (c) 2007 CodeSourcery.
 * Written by Paul Brook
 *
 * This code is licensed under the GPL.
 */

#include "qemu/osdep.h"
#include "hw/input/gamepad.h"
#include "hw/irq.h"
#include "ui/console.h"

/** COL733: Defines one button of Gamepad. **/
typedef struct {
    qemu_irq irq; // COL733: which interrupt does this button raise
    int keycode;   // COL733: What is the keycode of this button.
    uint8_t pressed;  // COL733: Is this button pressed?
} gamepad_button;

/** COL733: Defines the overall state of the gamepad. **/
typedef struct {
    gamepad_button *buttons;  // COL733: A linked list of all the buttons
    int num_buttons;  // COL733: Length of the linked list
} gamepad_state;

/* COL733: Initialize the virtual gamepad. */
void stellaris_gamepad_init(int n, qemu_irq *irq, const int *keycode)
{
    gamepad_state *s;
    int i;

    s = g_new0(gamepad_state, 1);  // COL733: Similar to calloc. Allocate 1
gamepad_state sized object and return its pointer.
    s->buttons = g_new0(gamepad_button, n);  // COL733: Allocate a linked
list of n gamepad_button sized objects. Return head of the linked list.
    for (i = 0; i < n; i++) {
        s->buttons[i].irq = irq[i];
        s->buttons[i].keycode = keycode[i];
    }
    s->num_buttons = n;
    qemu_add_kbd_event_handler(stellaris_gamepad_put_key, s); // COL733:
Register stellaris_gamepad_put_key function with QEMU’s keyboard event
handler. Whenever a keyboard button is pressed, QEMU will call the
registered function pointer with “opaque = s”.
}

static void stellaris_gamepad_put_key(void * opaque, int keycode)
{
    gamepad_state *s = (gamepad_state *)opaque;
    int i;
    int down = (keycode & 0x80) == 0;

    // Loop 1
    for (i = 0; i < s->num_buttons; i++) {
        if (s->buttons[i].keycode == keycode
                && s->buttons[i].pressed != down) {
            s->buttons[i].pressed = down;
            qemu_set_irq(s->buttons[i].irq, down);
        }
    }
}
```

a. [5 marks] What do you think “Loop 1” is trying to do?

The previous code is initiated by hw/arm/stellaris.c as follows:

```
static void stellaris_init(MachineState *ms, stellaris_board_info *board) {
    …..
    if (board->peripherals & BP_GAMEPAD) {
        qemu_irq gpad_irq[5];
        static const int gpad_keycode[5] = {0xc8, 0xd0, 0xcb, 0xcd, 0x1d};

        gpad_irq[0] = qemu_irq_invert(gpio_in[GPIO_E][0]); /* up */
        gpad_irq[1] = qemu_irq_invert(gpio_in[GPIO_E][1]); /* down */
        gpad_irq[2] = qemu_irq_invert(gpio_in[GPIO_E][2]); /* left */
        gpad_irq[3] = qemu_irq_invert(gpio_in[GPIO_E][3]); /* right */
        gpad_irq[4] = qemu_irq_invert(gpio_in[GPIO_F][1]); /* select */

        stellaris_gamepad_init(5, gpad_irq, gpad_keycode);
    }
   ….
}
```

b. [5 marks] This code sets up I/O emulation to emulate mouse up, left, down, and right as gamepad joystick's up, left, down, and right respectively and the left control button as gamepad's select. How would you modify the emulator so that it emulates keyboard presses of W, A, S, and D respectively as gamepad joystick's up, left, down, and right respectively and Return key as gamepad's select?

Here are the keycodes for all the keys: (From https://gist.github.com/rickyzhang82/8581a762c9f9fc6ddb8390872552c250

1	# Esc	9	# 8	211	# Delete	52	# .
59	# F1	10	# 9	207	# End	53	# /
60	# F2	11	# 0	209	# Page Down	54	# Shift Right
61	# F3	12	# -	58	# Caps Lock	43	# \
62	# F4	13	# =	30	# A	200	# Cursor Up
63	# F5	14	# Backspace	31	# S	29	# Ctrl Left
64	# F6	210	# Insert	32	# D	219	# Logo Left (-> Option)
65	# F7	199	# Home	33	# F	56	# Alt Left (-> Command)
66	# F8	201	# Page Up	34	# G	57	# Space
67	# F9	69	# Num Lock	35	# H	184	# Alt Right (-> Command)
68	# F10	15	# Tab	36	# J	220	# Logo Right (-> Option)
87	# F11	16	# Q	37	# K	221	# Menu (-> International)
88	# F12	17	# W	38	# L	157	# Ctrl Right
183	# PrintScrn	18	# E	39	# ;	203	# Cursor Left
70	# Scroll Lock	19	# R	40	# '	208	# Cursor Down
197	# Pause	20	# T	42	# Shift Left	205	# Cursor Right
41	# `	21	# Y	44	# Z		
2	# 1	22	# U	45	# X		
3	# 2	23	# I	46	# C		
4	# 3	24	# O	47	# V		
5	# 4	25	# P	48	# B		
6	# 5	26	# [	49	# N		
7	# 6	27	#]	50	# M		
8	# 7	28	# Return	51	# ,		